

Course 1

演算法：效率、分析與量級

Algorithms: Efficiency, Analysis, and Order

■ Outlines

◆ 本章重點

■ Algorithm

- Def. 與5個性質
- Pseudocode

■ The Importance of Developing Efficient Algorithms

■ Analysis of Algorithms

- Space complexity
- Time complexity

■ Asymptotic Notation (漸近式表示)

- Order
- $O, \Omega, \theta, o, \omega$

■ Using a Limit to Determine Order

■ Algorithm

◆ 通常在針對某一問題開發程式時，都會經過下列步驟：

Step 1: 明確**定義問題**

Step 2: 設計**演算法**，並評估其執行效率

Step 3: 撰寫**程式**，並加以測試

◆ **Example:** 計算大學入學考試中，某一單科分數之高標

■ **明確定義:** 計算所有考生在該科中**前25%成績**之平均。

■ **演算法:**

Step 1: 將所有考生英文成績**排序** (由高至低)

Step 2: 將排名在前面**1/4**的成績資料相加後，再除以**1/4**的人數

■ **撰寫程式:**...

- ◆ 當我們使用某種技巧解決一個問題時，會產生一種逐步執行的程序(**step-by-step procedure**)來解決問題，該種逐步執行的程序即稱為解決這個問題的**演算法**。
- ◆ Def:
 - 完成特定功能之有限個指令之集合。
 - 需滿足下列5個性質：
 - **Input**: 外界至少提供 ≥ 0 個輸入
 - **Output**: Algorithm至少產生 ≥ 1 個輸出結果
 - **Definiteness** (明確): 每個指令必須是Clear and Unambiguous
 - **Finiteness** (有限性): Algorithm在執行有限個步驟後，必定終止
 - **Effectiveness** (有效性): 用紙跟筆即可追蹤Algorithm中執行的過程及結果

Pseudocode (虛擬碼)



Note:

*One of the most common tools for defining algorithms is **pseudocode**, which is part **English**, part **Structured Code**.*

Example of Pseudocode

◆ 問題1: 搜尋問題

Example 1.2

The following is an example of a problem:

Determine whether the number x is in the list S of n numbers. The answer is yes if x is in S and no if it is not.

◆ 範例:

Example 1.4

An instance of the problem in [Example 1.2](#) is

$S = [10, 7, 11, 5, 13, 8]$, $n = 6$, and $x = 5$

The solution to this instance is, "yes, x is in S ."

◆問題演算法:

Algorithm 1.1: Sequential Search

Problem: Is the key x in the array S of n keys?

Inputs (parameters): positive integer n , array of keys S indexed from 1 to n , and a key x .

Outputs: *location*, the location of x in S (0 if x is not in S .)

```
void seqsearch(int n,
               const keytype S [ ],
               keytype x,
               index & location)

{
    location = 1;
    while (location <= n && S[location] != x)
        location ++;
    if (location > n)
        location=0;
}
```

No standard for pseudocode

- 可以寫得很像**英文敘述**
- 也可以寫得很像**程式語句**

■ 發展有效率演算法的重要性

◆ 不管電腦變得多快, 記憶體變得多便宜, 效率 (**Efficiency**) 仍然是設計演算法時最重要的考量.

◆ **排序問題 (Sorting problem):**

- 將一組資料以**遞增 (increasing)** 或以**遞減 (decreasing)** 的順序加以排列.

11, 7, 14, 1, 5, 9, 10

↓ **sort**

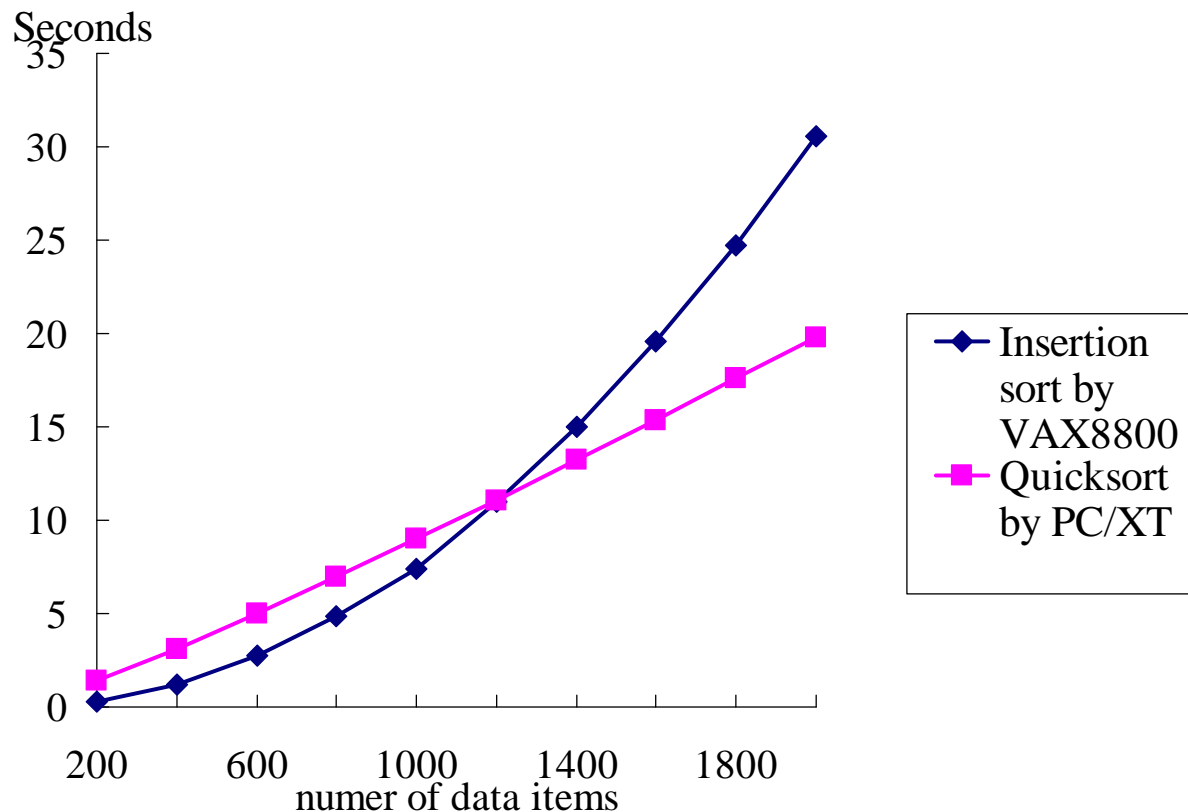
1, 5, 7, 9, 10, 11, 14

- 欲比較的兩個演算法:

- **Insertion sort (插入排序法):** $O(n^2)$
- **Quick sort (快速排序法):** $O(n \log n)$

■ 兩個演算法所安裝之系統:

- Quick Sort: **IBM PC/XT** (1983年產品, 以 Intel 8088 CPU為核心)
- Insertion Sort: **VAX8800** (DEC超級迷你電腦)



■ 演算法的分析

- ◆ 若要得知一個演算法解決一個問題有效率的程度，我們必須分析(**Analyze**)該演算法。
- ◆ 一個**Algorithm**的好壞，通常有兩個評估因子：
 - **Space (空間): Space Complexity**
 - 記憶體複雜度 (Memory Complexity)
 - **Time (時間): Time Complexity**

時間複雜度 (Time Complexity)

- ◆ 一般來說，對一個演算法進行時間複雜度分析就是求得：
 - 在每個不同的輸入大小 (**the size of the input**)之下，該演算法所執行的基本運算次數 (**how many times some basic operation is done**)。
- ◆ 分析方式：
 - 我們分析一個演算法的效率，是將該演算法在每個不同的輸入大小 之下，所執行的基本運算次數 設定成一個**函數 (Function** ; 或稱 **Time function, Complexity function**亦可) **$T(n)$** 。
 - **$T(n)$** 被定義為在大小為 n 的輸入範例下，某一個演算法所執行的基本運算次數。**(the number of times the algorithm does the basic operation for an instance of size n .)**

範例

每行指令的執行次數	程式
	<code>float sum(float list[], int n)</code>
	<code>{</code>
←	<code>int i;</code>
←	<code>float tempsum = 0;</code>
←	<code>for (i=0; i<n; i++)</code>
	<code>{</code>
←	<code>tempsum += list[i];</code>
	<code>}</code>
←	<code>return tempsum;</code>
	<code>}</code>

不可執行!!：就系統執行的角度而言，變數宣告只是在**Compile**時，在**M.M.**中建立一個空間，但不會產生相對應的執行碼!!除非在宣告的同時有指派一個初始值，指派的動作就會有相對應的執行碼產生以供程式執行時使用。

- ◆ 有許多被設計出來的演算法，當**輸入資料的情況不同**時，所分析出來的執行次數也有所不同。
- ◆ 因此，在分析這些演算法的執行次數時，可依據輸入資料的不同情況區分成三個**Case**來加以分析：
 - **The worst case (最差情況)**
 - 輸入大小為 n 的情況下，演算法執行基本運算次數的**最大值**。
 - $W(n)$: **worstcase time complexity**
 - **The best case (最佳情況)**
 - 輸入大小為 n 的情況下，演算法執行基本運算次數的**最小值**。
 - $B(n)$: **best-case time complexity**
 - **The average case (平均情況)**
 - 輸入大小為 n 的情況下，演算法執行基本運算次數的**平均值**。
 - $A(n)$: **average-case time complexity**

Worst-case time complexity analysis

◆ 例：循序搜尋法 (Sequential Search)

- 假設一定會搜尋到所要求的item

Analysis of Algorithm 1.1 Worst-Case Time Complexity (Sequential Search)

Basic operation: the comparison of an item in the array with x .

Input size: n , the number of items in the array.

The basic operation is done at most n times, which is the case if x is the last item in the array or if x is not in the array. Therefore,

$$W(n) = n.$$

Best-case time complexity analysis

◆ 假設一定會搜尋到所要求的item

Analysis of Algorithm 1.1 Best-Case Time Complexity (sequential Search)

Basic operation: the comparison of an item in the array with x .

Input size: n , the number of items in the array.

Because $n \geq 1$, there must be at least one pass through the loop. If $x = S[1]$, there will be one pass through the loop regardless of the size of n . Therefore,

$$B(n) = 1.$$

Average-case time complexity analysis

◆ 假設一定會搜尋到所要求的item

Analysis of Algorithm 1.1 Average-Case Time Complexity (Sequential Search)

Basic operation: the comparison of an item in the array with x .

Input size: n , the number of items in the array.

We first analyze the case in which it is known that x is in S , where the items in S are all distinct, and where we have no reason to believe that x is more likely to be in one array slot than it is to be in another. Based on this information, for $1 \leq k \leq n$, the probability that x is in the k th array slot is $1/n$. If x is in the k th array slot, the number of times the basic operation is done to locate x (and, therefore, to exit the loop) is k . This means that the average time complexity is given by

$$A(n) = \sum_{k=1}^n \left(k \times \frac{1}{n} \right) = \frac{1}{n} \times \sum_{k=1}^n k = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}.$$

The third step in this quadruple equality is derived in [Example A.1](#) of [Appendix A](#). As we would expect, on the average, about half the array is searched.

- ◆ 對於上述時間複雜度分析，我們較常進行最差情況與平均情況的分析，而不是最佳情況分析。
 - **平均情況**分析是很有用的，這是因為它告訴我們：一個演算法在有許多不同的輸入狀況時，總共所花費的時間為何。
 - Ex: A **sorting algorithm** that was used repeatedly to sort all possible inputs.
 - **最差情況**分析是很有用的，這是因為它告訴我們：一個演算法**執行時間的上限為何**。
 - Ex: A system that **monitored a nuclear power plant**
 - 相對來說，最佳情況分析的用處實在不大。
- ◆ 通常，平均情況比最差情況要難分析。(∴ **Worst Case**最常被討論)

◆ $T(n)$ 被稱為所有情況時間複雜度 (**Every-case time complexity**), 這是因為:

- 對大小為 n 的所有輸入範例, 其基本運算都是執行同樣的次數。也就是說, 此時間複雜度的計算並不考慮輸入資料的各種不同情況, 僅以**資料量**為主要考量依據。

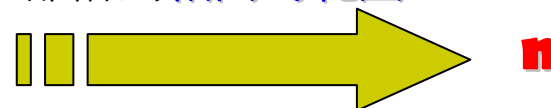
◆ 求得 $T(n)$ 的過程稱為**所有情況時間複雜度分析 (Every-case time complexity analysis)**。

■ 量級 (Order)

- ◆ 前面所提到之基本運算的執行次數，可利用**漸近式符號**(或稱**量級**, **Order**)予以表示。
- ◆ 主要原因：
 - 有些程式間**實際執行次數**看似不同，但是在電腦上執行的**效能**卻差不多。

for i=1 to n do a=(b+c)/d+e;	for i=1 to n do T1=b+c; T2=T1/d; a=T2+e;
⇒ T(n)=n	⇒ T(n)=3n

都落到**相同的範圍**



- **指令有的簡單，有的複雜**。如：
 - 浮點數運算比整數運算難
 - 除法和加法運算的複雜性不同

◆ 因此, 分析一個演算法的執行次數或時間複雜度, 常使用**Order** (**量級**, 或稱**等級**)的觀念。

■ 將具有不同但近似之執行次數的一些演算法歸納到**相同的時間等級**之中。

◆ 一般演算法分析常見的**Order** 大小排序如下:

$$1(\text{或常數}) < \log n < n < n \log n < n^2 < n^3 < 2^n < 3^n < (n/2)^{n/2} \leq n!$$

■ **1**: 表示敘述的基本運算的次數是**固定的(constant)**,

■ **n**: 表示敘述運算次數之**Order**呈**線性成長**。

■ **n²**: 表示敘述運算次數之**Order**呈**二次多項式曲線成長**。

■ **n³**: 表示敘述運算次數之**Order**呈**三次多項式曲線成長**。

◆ 課本上的 $\lg n = \log_2 n$

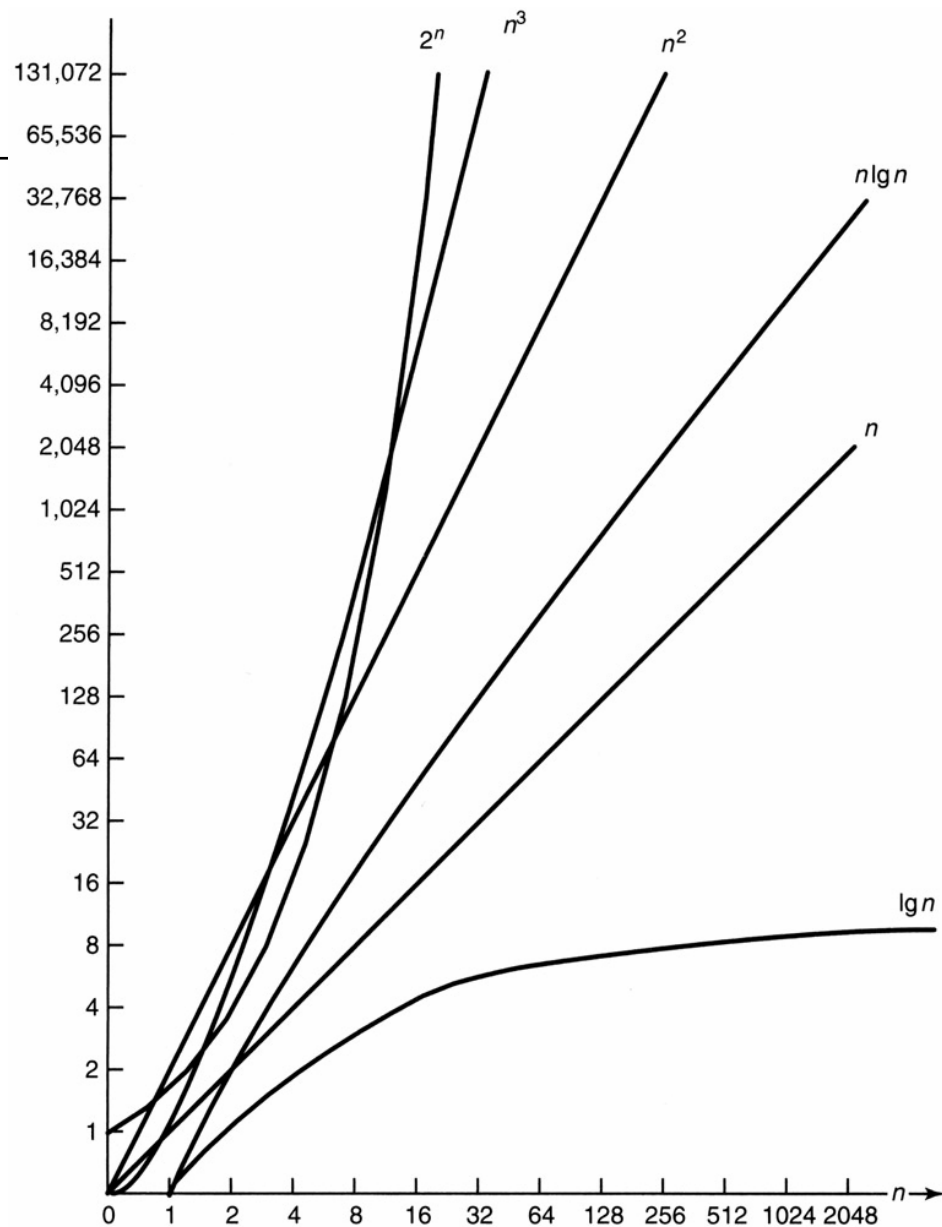
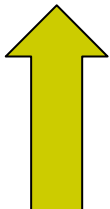


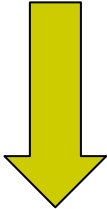
Figure 1.3: • Growth rates of some common complexity functions

- ◆ 研究演算法是為了**正確有效率**地解決問題。

尚可接受



$f(n) \setminus n$	10	10^2	10^3
$\log_2 n$	3.3	6.6	10
n	10	10^2	10^3
$n \log_2 n$	0.33×10^2	0.7×10^3	10^4
n^2	10^2	10^4	10^6
2^n	1024	1.3×10^3	$> 10^{100}$
$n!$	3^6	$> 10^{100}$	$> 10^{100}$



需要改進

以直覺的方式介紹Order

◆ 有下列三個代表演算法執行時間的時間函數：

■ $5n^2$

■ $5n^2 + 100$

■ $0.1n^2 + n + 100$



$\Theta(n^2)$

Θ 僅表示某一個演算法分析方式的表示符號

⇒ 可稱這些函數為平方時間(**Quadratic term**).

◆ 某個演算法的時間複雜度在 $\Theta(n^2)$ 中，該演算法就被稱為平方時間演算法 (**quadratic-time algorithm**)。

◆ 通常，一個演算法的複雜度必須為 $\Theta(n \lg n)$ 或更好，我們才會認為這個演算法能在可忍受的時間內處理輸入大小非常大的範例。

以嚴謹的方式介紹**Order**

◆ 共有五種漸近式表示方法：

- **Big-O** (**O**)
- **Omega** (**Ω**)
- **Theta** (**θ**)
- **Small-O** (**o**)
- **Small-Omega** (**ω**)

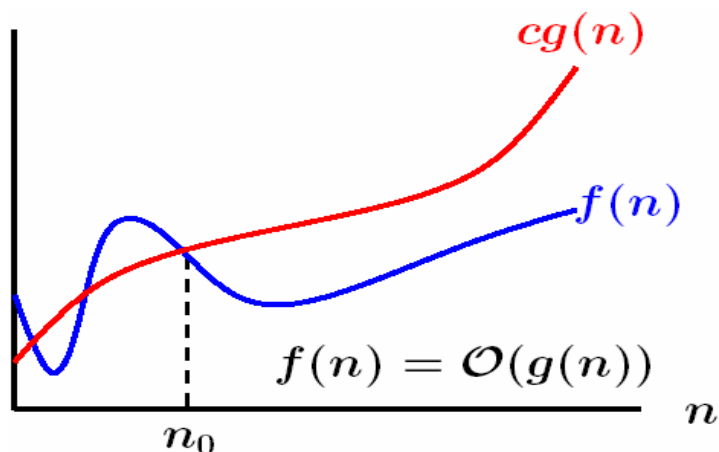
Big-O (O)

◆ Upper bound of $f(n)$.

- 在一個多項式中，取**最大項**當成是理論上限，即程式執行時**花費時間的成長率**。

◆ Definition:

- $f(n) = O(g(n))$ if and only if 存在兩正數 c 和 n_0 , 使得 $f(n) \leq c \times g(n)$, for all $n \geq n_0$.



- ◆ n ：輸入資料量大小。
- ◆ $f(n)$ ：在理想狀況下，程式在電腦中的**指令實際執行次數**。
- ◆ $g(n)$ ：執行時間的**成長率**。

以定義來說明範例

◆ $f(n) = 3n+2$, 則 $f(n) = O(n)$.

■ $f(n) = O(n)$ if and only if 存在兩正數 $c = \underline{\quad}$ 和 $n_0 = \underline{\quad}$, 使得 $f(n) \leq c \times g(n)$, for all $n \geq n_0$.

- 先決定 c 的值, 鎖定 $f(n)$ 中的最大項之值(即: $3n$), c 只要比該項的常數值 **大1** 即可!! 再由 c 去推 n_0 .
- $3n+2 \leq 4n \Rightarrow n \geq 2$.

◆ $f(n) = 5n^2+3n+2$, 則 $f(n) = O(n^2)$.

■ $f(n) = O(n^2)$ if and only if 存在兩正數 $c = \underline{\quad}$ 和 $n_0 = \underline{\quad}$, 使得 $f(n) \leq c \times g(n)$, for all $n \geq n_0$.

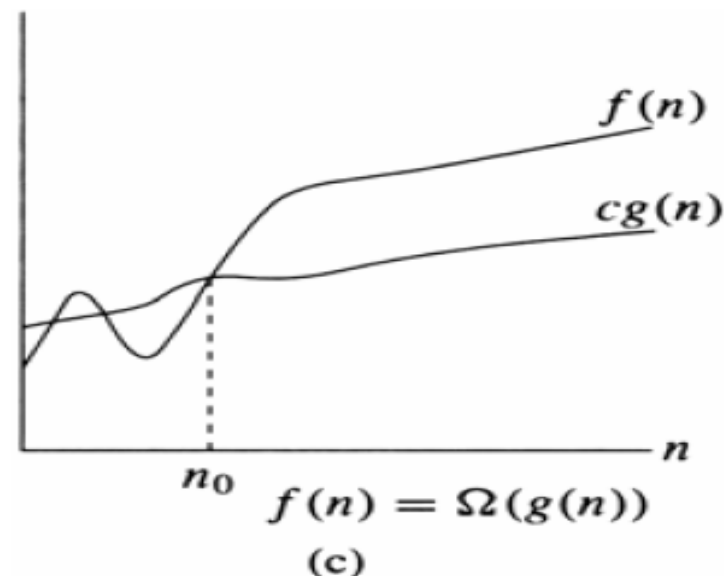
- 先決定 c 的值, 鎖定 $f(n)$ 中的最大項之值(即: $5n^2$), c 只要比該項的常數值 **大1** 即可!! 再由 c 去推 n_0 .
- $5n^2+3n+2 \leq 6n^2 \Rightarrow 3n+2 \leq n^2 \Rightarrow$ 取 $n_0 = 4$.

Omega (Ω)

◆ **Lower bound** of $f(n)$.

◆ **Definition:**

- $f(n) = \Omega(g(n))$ if and only if 存在兩正數 c 和 n_0 , 使得 $f(n) \geq c \times g(n)$, for all $n \geq n_0$.



lower bound

以定義來說明範例

◆ $f(n) = 3n+2$, 則 $f(n) = \Omega(n)$.

■ $f(n) = \Omega(n)$ if and only if 存在兩正數 $c = \underline{\quad}$ 和 $n_0 = \underline{\quad}$, 使得 $f(n) \geq c \times g(n)$, for all $n \geq n_0$.

- 先決定 c 的值, 鎖定 $f(n)$ 中的最大項之值(即: $3n$), c 只要與該項的常數值相同即可!!再由 c 去推 n_0 。
- $3n+2 \geq 3n \Rightarrow 2 \geq 0$ (恒真, 故 n_0 可任取!)

◆ $f(n) = 5n^2+3n-2$, 則 $f(n) = \Omega(n^2)$.

■ $f(n) = \Omega(n^2)$ if and only if 存在兩正數 $c = \underline{\quad}$ 和 $n_0 = \underline{\quad}$, 使得 $f(n) \geq c \times g(n)$, for all $n \geq n_0$.

- 先決定 c 的值, 鎖定 $f(n)$ 中的最大項之值(即: $5n^2$), c 只要與該項的常數值相同即可!!再由 c 去推 n_0 。
- $5n^2+3n-2 \geq 5n^2 \Rightarrow 3n-2 \geq 0 \Rightarrow$ 取 $n_0 = 2/3$ 。

Theta (θ)

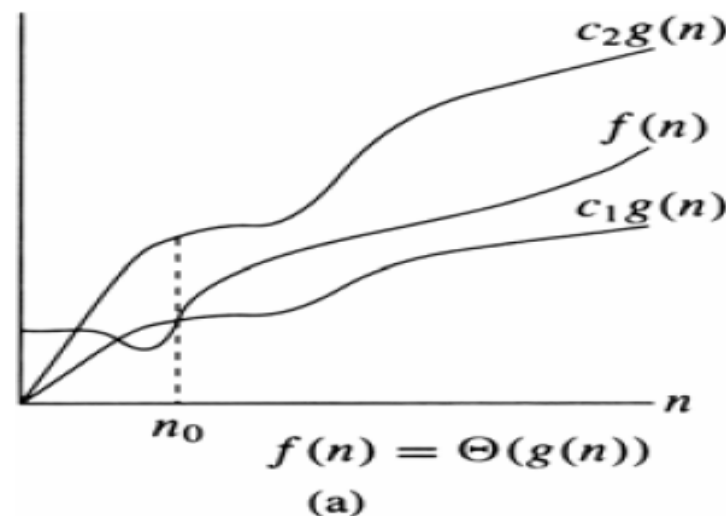
◆ 較 O 與 Ω 精確.

◆ Definition:

■ $f(n) = \theta(g(n))$ if and only if 存在三正數 c_1 , c_2 和 n_0 , 使得

$$c_1 \times g(n) \leq f(n) \leq c_2 \times g(n),$$

for all $n \geq n_0$.



tight bound

以定義來說明範例

◆ $f(n) = 3n+2$, 則 $f(n) = \theta(n)$.

■ $f(n) = \theta(n)$ if and only if 存在三正數 $c_1 = __, c_2 = __$ 和 $n_0 = __$, 使得 $c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$, for all $n \geq n_0$.

○ 用先前決定 O 與 Ω 中 c 值的方式分別得到 c_1 與 c_2 即可!!再由 c_1 與 c_2 去推 n_0 。

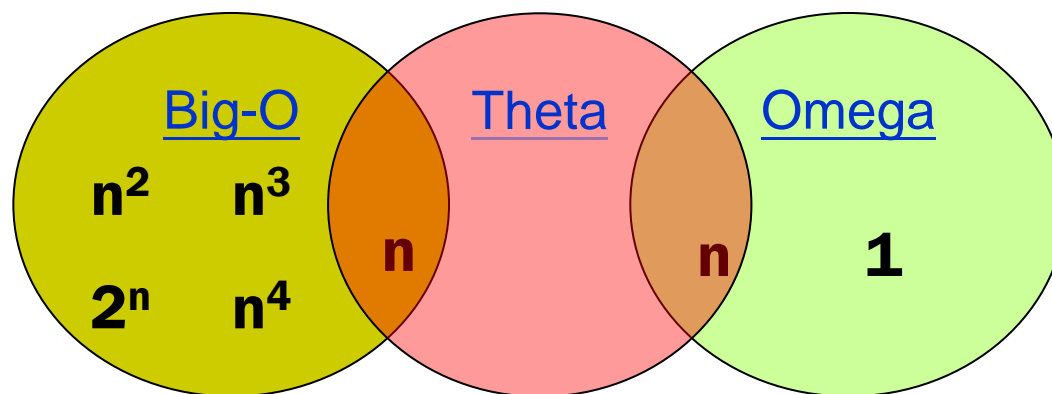
○ $3n+2 \leq 4n$ (用 $f(n) \leq c_2 \times g(n)$ 來找) $\Rightarrow n \geq 2$ 。

※ If $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$, 則 $f(n) = \theta(g(n))$.

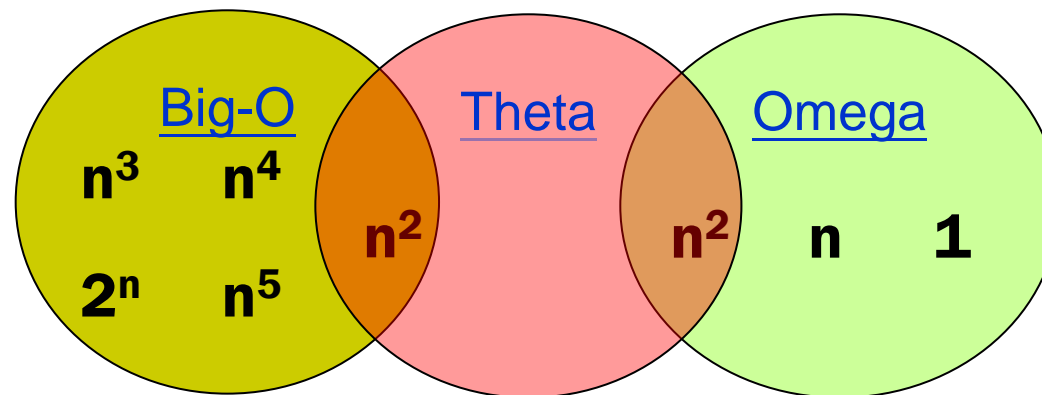
Big-O, Omega與Theta的關係

◆ 以 $f(n) = 3n+2$ 與 $f(n) = 5n^2+3n+2$ 為例:

■ $3n+2$:



■ $5n^2+3n+2$:



Small-O (o)

◆ Definition:

- $f(n) = o(g(n))$ if and only if 對任何正數 c , 會存在一個正數 n_0 , 使得 $f(n) < c \times g(n)$, for all $n \geq n_0$.

◆ 與 Big-O 的比較:

- $f(n) = O(g(n))$ if and only if 存在兩正數 c 和 n_0 , 使得 $f(n) \leq c \times g(n)$, for all $n \geq n_0$.
- 例如:
 - $2n = o(n^2)$
 - $2n^2 \neq o(n^2)$, 但是 $2n^2 = O(n^2)$ 。

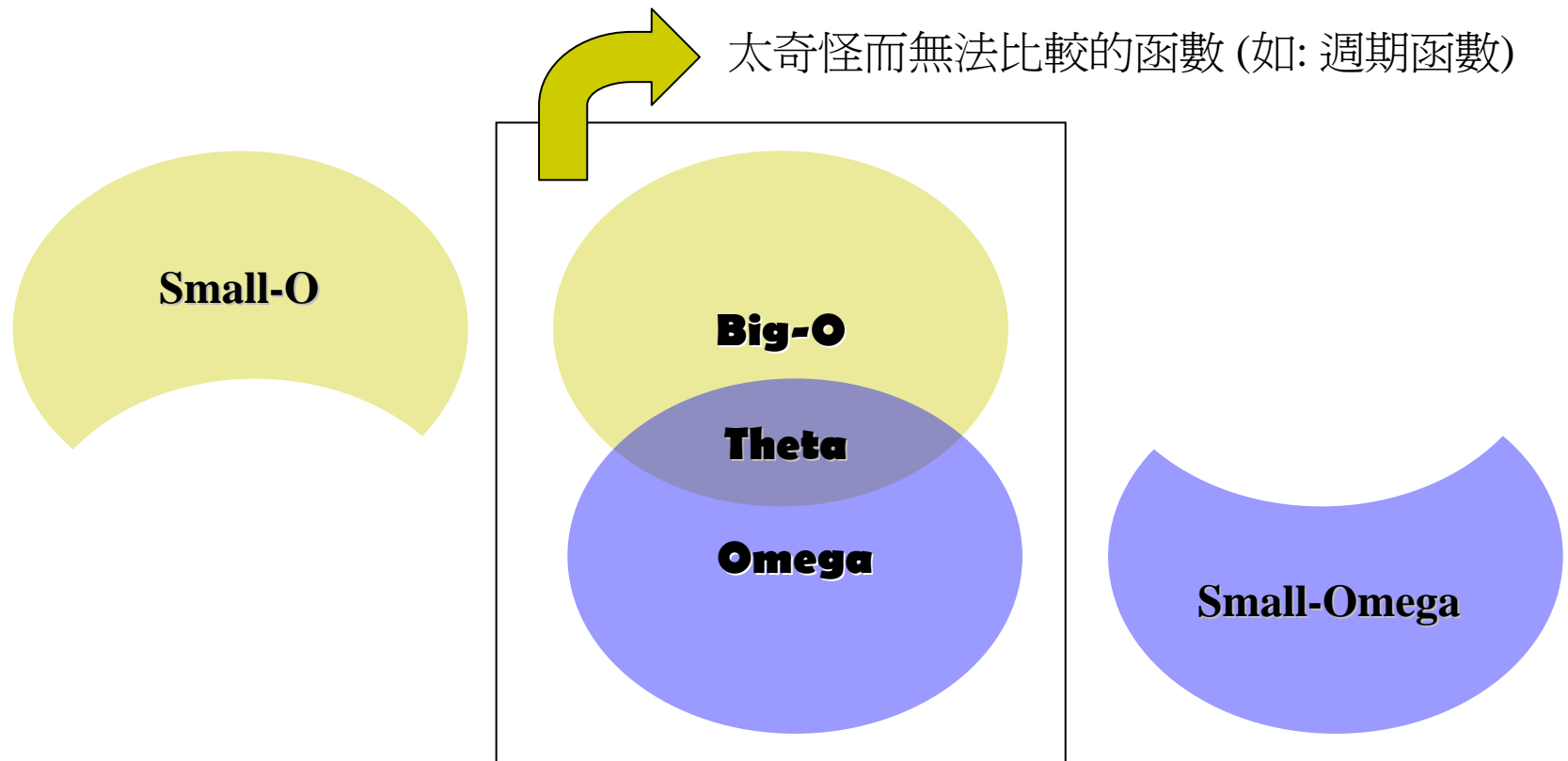
Small-Omega (ω)

◆ Definition:

- $f(n) = \Omega(g(n))$ if and only if 對任何正數 c , 會存在一個正數 n_0 , 使得 $f(n) > c \times g(n)$, for all $n \geq n_0$.

◆ 與Omega的比較:

- $f(n) = \Omega(g(n))$ if and only if 存在兩正數 c 和 n_0 , 使得 $f(n) \geq c \times g(n)$, for all $n \geq n_0$.
- 例如:
 - $5n^2 + 3n + 2 = \omega(n)$
 - $5n^2 + 3n + 2 \neq \omega(n^2)$, 但是 $5n^2 + 3n + 2 = \Omega(n^2)$



■ 使用極限(Limit)來決定Order

- ◆ 若我們想利用極限的方式證明 $f(n) = \Theta(g(n))$ (Θ 僅表示某一個演算法分析方式的表示符號), 則:

Note

1. 若 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, 則 $f(n) = o(g(n))$ 。因此必保證 $f(n) = O(g(n))$ 。
2. 若 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = L > 0$, 則 $f(n) = \Theta(g(n))$ 。
3. 若 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$, 則 $f(n) = \omega(g(n))$ 。因此必保證 $f(n) = \Omega(g(n))$ 。

符號	定義	極限判斷法
Big-O (O)	$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 > 0, \ni f(n) \leq cg(n), \forall n \geq n_0.$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
Small-O (o)	$f(n) = o(g(n)) \Leftrightarrow \forall c > 0, \exists n_0 > 0, \ni f(n) < cg(n), \forall n \geq n_0.$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
Omega (Ω)	$f(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 > 0, \ni f(n) \geq cg(n), \forall n \geq n_0.$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
Small-Omega (ω)	$f(n) = \omega(g(n)) \Leftrightarrow \forall c > 0, \exists n_0 > 0, \ni f(n) > cg(n), \forall n \geq n_0.$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
Theta (θ)	$f(n) = \theta(g(n)) \Leftrightarrow \exists c_1, c_2, n_0 > 0, \ni c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0.$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = L$

◆ 羅必達法則 (L' Hospital Rule)

Ex: Determine if $f(n) = \begin{matrix} \Omega \\ \theta \\ o \end{matrix} (g(n))$ or neither

1. $f(n) = \log_5 4n$, $g(n) = \log_4 5n$

Sol:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\log_5 4n}{\log_4 5n} = \lim_{n \rightarrow \infty} \frac{\frac{\ln 4n}{\ln 5}}{\frac{\ln 5n}{\ln 4}} = \lim_{n \rightarrow \infty} \frac{\frac{\ln 4 + \ln n}{\ln 5}}{\frac{\ln 5 + \ln n}{\ln 4}} = \lim_{n \rightarrow \infty} \frac{\ln 4 \times (\ln 4 + \ln n)}{\ln 5 \times (\ln 5 + \ln n)}$$

$$\lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)} = \lim_{n \rightarrow \infty} \frac{0 \times (\ln 4 + \ln n) + \ln 4 \times (0 + \frac{1}{n})}{0 \times (\ln 5 + \ln n) + \ln 5 \times (0 + \frac{1}{n})} = \frac{\ln 4}{\ln 5} = \text{常數} > 0$$

$\therefore f(n) = \theta(g(n)).$

2. $f(n) = 1.1^{0.1n}$, $g(n) = n^3$

Sol:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\log f(n)}{\log g(n)} = \lim_{n \rightarrow \infty} \frac{0.1n \times \log 1.1}{3 \times \log n}$$

$$\lim_{n \rightarrow \infty} \frac{(\log f(n))'}{(\log g(n))'} = \lim_{n \rightarrow \infty} \frac{0.1 \times \log 1.1}{3 \times \frac{1}{n \ln 10}} = \lim_{n \rightarrow \infty} \frac{0.1 \times \log 1.1 \times n \ln 10}{3} = \infty$$

$\Rightarrow \log f(n) = \omega(\log g(n))$ ($\Omega(\log g(n))$ 亦可成立)

$\Rightarrow f(n) = \omega(g(n))$ ($\Omega(g(n))$ 亦可成立)

3. $f(n) = n \cot(n)$, $g(n) = n \tan(n)$

Sol:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n \cot(n)}{n \tan(n)} = \lim_{n \rightarrow \infty} \frac{\cot(n)}{\tan(n)} \quad \text{其值在 } -\infty \sim \infty \text{ 間變動}$$

\therefore Neither.

Note

1. Log的底和Complexity無關
2. a. 若 $\log f(n) = o(\log g(n))$, 可保証 $f(n) = o(g(n))$
b. 若 $\log f(n) = \omega(\log g(n))$, 可保証 $f(n) = \omega(g(n))$
c. 若 $\log f(n) = \theta(\log g(n))$, 不可保証 $f(n) = \theta(g(n))$
3. 兩函數的Complexity有可能無法比較 (Ex: 週期函數)

◆ Note 2說明 (反例)

■ $f(n) = 2^{\lg n}$, $g(n) = n^2$ ($\lg = \log_2$)

Sol:

$$\lg f(n) = \lg n \times \lg 2$$

$$\lg g(n) = 2 \lg n$$

$$\therefore \lg f(n) = \theta(\lg g(n))$$

???

$$\because f(n) = 2^{\lg n} = n, \\ g(n) = n^2$$

※課後練習※

◆ 課文內容:

- **Example 1.7, Example 1.9, Example 1.10, Example 1.11 Example 1.12, Example 1.13, Example 1.15, Example 1.17, Example 1.18, Example 1.19, Example 1.24, Example 1.25, Example 1.26, Example 1.27, Example 1.28.**

◆ 習題:

- **15, 16, 17, 18** (上學期資結Ch. 1有教), **24**
- **21** 請自行參考洪捷p.1-10定理3及例題7~10

◆ 洪捷演算法Ch. 1:

- **例 1~6**
- **精選範例 1, 2**

補充

補 1: 常用的數學式子

$$(1) \text{ 等差數列: } \sum_{i=1}^n i = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$(2) \text{ 等比數列: } \sum_{i=0}^n r^i = r^0 + r^1 + r^2 + \dots + r^n = \frac{r^{n+1} - r^0}{r - 1}$$

$$(3) \quad C_m^n = \frac{n!}{m!(n-m)!}, \quad P_m^n = \frac{n!}{m!}$$

$$(4) \quad \sum_{i=1}^n i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

-
- ① $\log \log n = \log (\log n)$
 - ② $\log^k n = (\log n)^k$
 - ③ $a = b^{\log_b a}$
 - ④ $\log_c ab = \log_c a + \log_c b$
 - ⑤ $\log_c a/b = \log_c a - \log_c b$
 - ⑥ $\log_b a^n = n \log_b a$
 - ⑦ $\log_b a = \log_c a / \log_c b$
 - ⑧ $\log_b a = 1 / \log_a b$
 - ⑨ $\log_b (1/a) = \log_b a^{-1} = -\log_b a$
 - ⑩ $a^{\log_b c} = c^{\log_b a}$

補 2: 輸入大小(the size of the input)的合理測量

- ◆ 在前面正課所提及之一般的演算法複雜度分析當中，我們通常將“**輸入到演算法中的資料量**”視為**輸入大小 (Input Size)**。
 - 從 n 筆資料中，搜尋一筆user想要得到的資料
 - 對 n 筆資料做由小到大的排序工作
 - 將一個graph輸入到某演算法做處理
- ◆ 有時候對於稱呼某個參數為輸入大小要非常小心

◆ 定義:(9.2節)

- 對於一個給定的演算法, 輸入大小(**Input Size**)為該演算法用來表示所輸入之資料的字元數 (符號數)

◆ 範例:

- 某一演算法有n個整數型態的輸入資料, 其中最大值為**31**, 資料編碼方式為二進位編碼。
 - 每一筆資料的字元數(符號數, 位元數; 即用來編碼的符號數目)為 $\lfloor \log_2 31 \rfloor + 1 = 5$
 - 該演算法的Input Size為5n

- ◆ 某一演算法有n個輸入資料, 其中最大值為**L**。若資料編碼方式為k進位編碼, 則該演算法的Input Size大約為:

$$n \times \lfloor \log_k L \rfloor$$

補 3: 補充考題

◆ **Ex 1: Show the following equality is incorrect**

(91交大)

$$n^2/\log n = \theta(n^2)$$

Sol 1: (以極限來做)

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\frac{n^2}{\log n}}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{\log n} = 0,$$
$$\therefore \frac{n^2}{\log n} = o(n^2) \text{ or } O(n^2), \text{ not } \theta(n^2).$$

Sol 2: (以定義來做)

- θ 需同時符合 $n^2/\log n = O(n^2)$ 和 $n^2/\log n = \Omega(n^2)$ 。(夾擠法)
- $n^2/\log n = O(n^2)$ 成立, 証明方式請依上課所教之過程自行練習。
- 假設 $n^2/\log n = \Omega(n^2)$ 為真, 則:

$$\Leftrightarrow \exists c > 0 \text{ 與 } n_0 > 0, \exists n^2/\log n \geq c \times n^2, \forall n \geq n_0$$

$$\Leftrightarrow 1/\log n \geq c, \forall n \geq n_0$$

但是, 當 $n \rightarrow \infty$ 時, $1/\log n \rightarrow 0$ 。然而 c 是一個大於 0 的正數, \therefore 我們找不到一個 $c > 0$ 。(矛盾)

$$\therefore n^2/\log n \neq \Omega(n^2)$$

- $n^2/\log n = \theta(n^2)$ 不成立!!!